Acnet Cycle Data Access

Working with SSDNs

Mon, Nov 29, 2004

Acnet uses the RETDAT protocol to collect device data from front ends. In order to collect the recently-supported Cycle data, one should use listype 88 at the lowest level. This can be done with an SSDN, but since the Acnet reading property is the only one suitable for any kind of ordinary analog data, it implies using a different device name for access to Cycle data, assuming that the device has already been installed with an SSDN for its reading property that specifies listype 0 to access the analog channel reading from the ADATA table. This note explores how to access Cycle data without creating a new device name.

One might think to do this by specifying a length (#bytes requested) of more than 2 bytes, which is the usual size of an analog channel reading. But this option has already been usurped for accessing waveform data for ordinary channels, at least those that have a CINFO table entry indicating that a waveform exists for the channel. Of course, in the absence of a CINFO entry for a given channel, one could interpret a request for more than 2 bytes of data as meaning that Cycle data should be returned. But there can easily be cases in which the waveform option makes sense and should continue to make sense.

What FTD values make sense for accessing Cycle data? One could imagine collecting such data at a rate as high as 15 Hz, since that would allow comparing it with data collected from other nodes and thereby nailing down on which 15 Hz accelerator cycle the data reading was measured. But Vax client nodes do not make 15 Hz data access reliable yet. When the Java support is in place, it will be possible to make such access reliable, and the cycle counter returned with the Cycle data can allow the Java requester to properly correlate such readings with those received from other front ends. The answer to the question is that any periodic FTD should be usable to obtain Cycle data, although practical limit of 2 seconds might be the least frequent rate, since the Cycle table data memory is limited to the last 32 cycles. If the request is event-based, then a reply is returned that includes data on all the recent cycles specified by the buffer size, including the present cycle reading. A string of clock events might provide some considerable overlap of data, but the cycle# tags can sort this out.

Use of a nonzero offset, except for those that result in internal modification of the ident portion of the SSDN, makes no sense for accessing Cycle data. Any return of Cycle data includes all the data that can fit into the user buffer, including the 2-byte cycle counter.

For the first implementation, the format of reply data for a Cycle data request starts with the cycle counter. If one asks for 4 bytes of data, the cycle counter is first followed by the single reading that is the reading value sampled from the current data pool. But suppose that the order is reversed, so that the cycle counter occurs last. If one asks for 2 bytes, one would get the present cycle reading. If one asks for 4 bytes, one would get the present cycle reading in the first word following by the cycle counter in the second word. Asking for 6 bytes, one would get the present cycle reading in the first word, the previous cycle reading in the second word, and the cycle counter as the third word. If one asked for 32 bytes, one would get the last 15 cycles of readings, present cycle first, and ending with the cycle counter. Perhaps the cycle counter should in this case refer to the present cycle, for which the reading is always the first word in the reply.

This still does not solve the waveform access ambiguity. Should we assume that waveform access always involves requesting a suitably large number of bytes, more than the Cycle table is likely to hold? Right now, one can even ask for a single word out of a waveform by using the offset word to specify a byte offset into the waveform. But one cannot ask for the first 2 bytes of the waveform by specifying an offset of zero, since that is interpreted as asking for the `ADATA` reading of the present cycle. One often might ask for the waveform data using offset = 0. But we may be able to assume that for such a case, at least 34 words of waveform data is sought, so that the request would be for at least 68 bytes. The largest Cycle data request that makes sense, given 32 cycles of 15 Hz history, is 66 bytes, which allows room for all 32 samples plus the cycle counter word. If we can assume that any request for a waveform that specifies a zero offset will seek at least 68 bytes, this would allow us to detect a request for Cycle data. A user would therefore be limited in waveform requests that ask for the start of the waveform, but would not be so limited in accessing portions of a waveform, in which the offset would be nonzero. One could still pick out an arbitrary word in a waveform by specifying the required nonzero offset.

Examining an `SSDN` in which the listype number is 0, if the request is for at least 4 bytes, and the offset is 0, one must consider the possibility that Cycle data is being sought. If there is no `CINFO` entry, then Cycle data it is. If there is a waveform specified by a `CINFO` entry, examine the length sought. If it is less than 68 bytes, then it is Cycle data; otherwise, it is waveform data. To retrieve the Cycle data, modify the listype from 0 to 88 internally. This should result in Cycle data returned in the reply. This testing should precede any testing for time-stamped data at 7.5 Hz. One should be able to get Cycle data at 7.5 Hz without causing confusion.

As to the idea of returning Cycle data in a new way, with the cycle counter at the end, how should this be decided? What advantage is there for either format? Does it matter?

Using listype 88 in the new format, a request for 2 bytes of data will yield the same value was is done by listype 0. If more than 2 bytes is requested, the present reading will still be in the first two bytes, and if more than 4 bytes is requested, the words following the first word will be readings that were captured on ever-earlier cycles. The cycle counter is always found at the end of the buffer, and it refers to the present cycle, for which the reading is found in the first word.

Acnet application support

Because dealing with the raw replies that arrive at different times from different devices can be messy, it may be helpful for an easier application interface to be supplied. Suppose one specifies a list of devices for which such readings are sought, along with a number of cycles of history that is desired. The `CycleOpen` function called with such a request returns an index number that can be used for subsequent calls to other functions.

One other function can be one for which a cycle# is given, asking for the next cycle# for which all the data is present for all devices in the original list. It might be called `CycleNext`. Another function can be called specifying such a cycle# and a buffer, and if all data is present for that cycle#, the buffer is filled with the values. It might be called `CycleData`. If desired, the previous two functions might be combined as `CycleNextData`. Other functions could be added as needed. Here are some examples of calls to such functions:

```
cycIndx = CycleOpen(nCycles, nDevices, devices);

cycStat = CycleNext(cycIndx, &cycNum);
cycStat = CycleData(cycIndx, cycNum, buffer);

cycStat = CycleNextData(cycIndx, buffer);

cycStat = CycleClose(cycIndx);
```

Note that in principle, the application support package could keep more than 32 cycles worth of Cycle data history. It may also be useful, though, to include a parameter to `CycleOpen` that sets the period of the Cycle data request, since this will affect the time that application processing lags the real-time accelerator process. (If Cycle data is returned at 1 Hz, the processing of that data can be as much as 1 second behind.)

Such a set of routines as those sketched out here makes available, in effect, a 15 Hz correlated data pool for an application to use. Any device that is part of the front end 15 Hz analog data pool is eligible for inclusion in this scheme.